Pipelines: The T-Mobile Way

Using Templates to Automate the CI/CD Process

P&T
PRODUCT & TECHNOLOGY

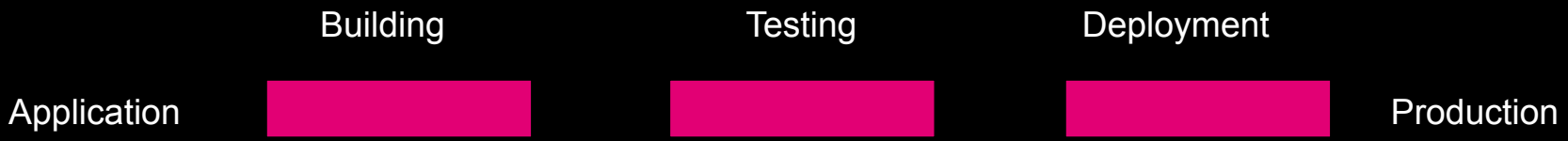
Objective

Students will build a pipeline the T-Mobile way using templates

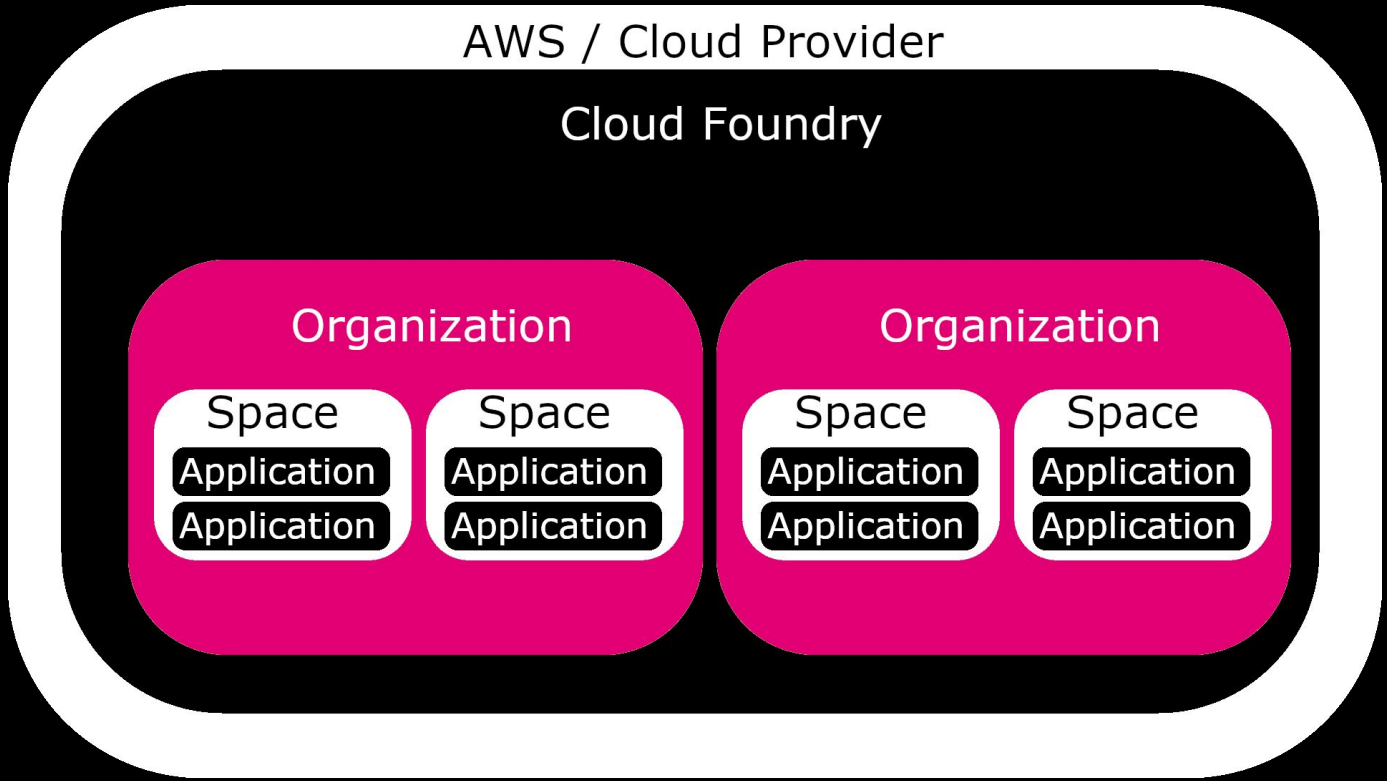
What is CI/CD?

Why do we care about CI/CD?

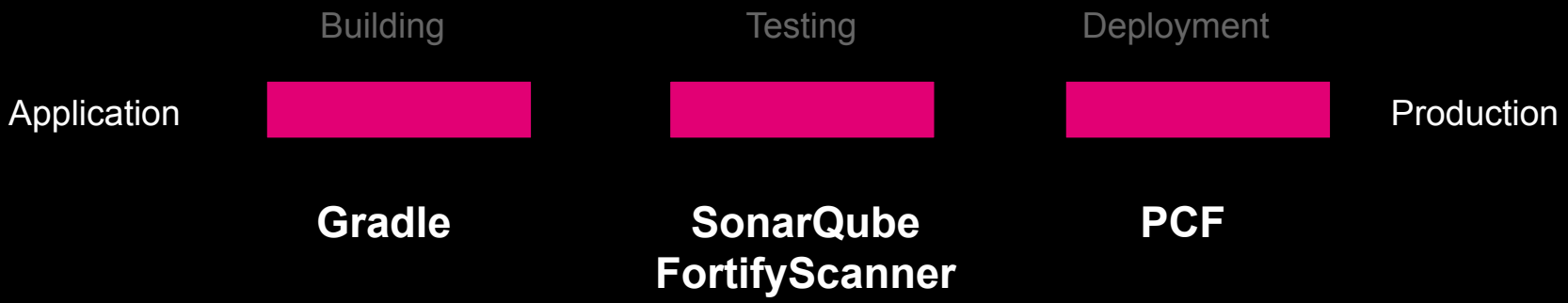
Pipelines



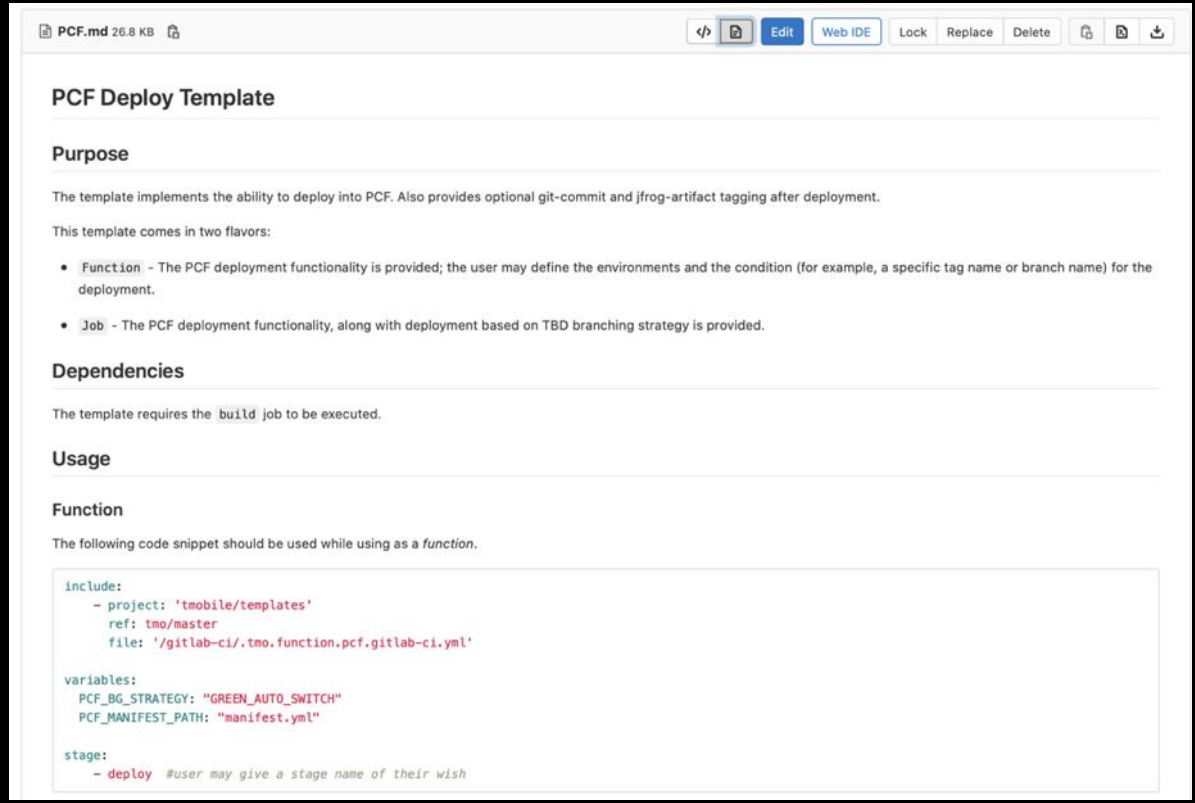
Pivotal Cloud Foundry



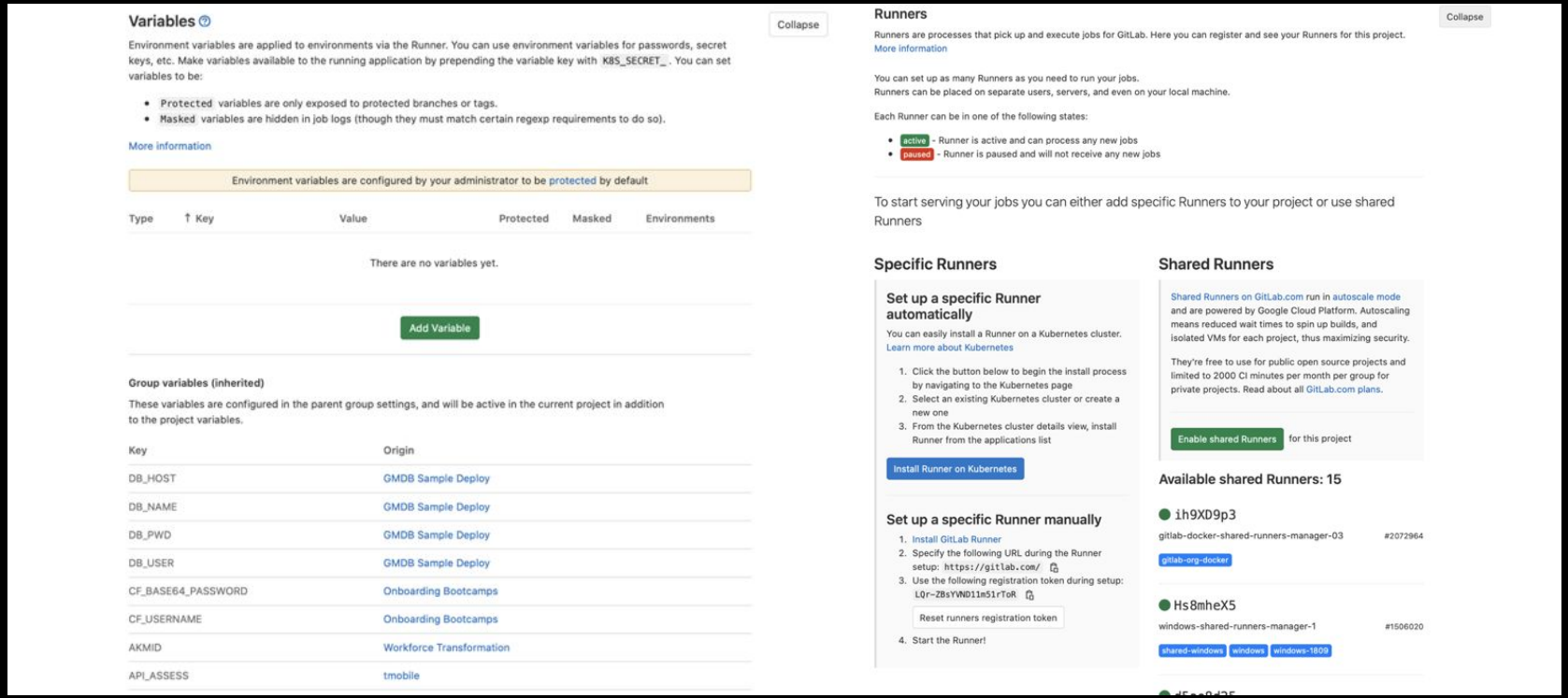
Templates


Building
Testing
Deployment
Application
Production
Gradle
SonarQube
FortifyScanner
PCF

Documentation Examples



Gitlab CI/CD



Variables

Collapse

Environment variables are applied to environments via the Runner. You can use environment variables for passwords, secret keys, etc. Make variables available to the running application by prepending the variable key with `KBS_SECRET_`. You can set variables to be:

- `Protected` variables are only exposed to protected branches or tags.
- `Masked` variables are hidden in job logs (though they must match certain regexp requirements to do so).

More information

Environment variables are configured by your administrator to be protected by default

Type	↑ Key	Value	Protected	Masked	Environments

There are no variables yet.

Add Variable

Group variables (inherited)

These variables are configured in the parent group settings, and will be active in the current project in addition to the project variables.

Key	Origin
DB_HOST	GMDB Sample Deploy
DB_NAME	GMDB Sample Deploy
DB_PWD	GMDB Sample Deploy
DB_USER	GMDB Sample Deploy
CF_BASE64_PASSWORD	Onboarding Bootcamps
CF_USERNAME	Onboarding Bootcamps
AKMID	Workforce Transformation
API_ASSESS	tmobile

Runners

Collapse

Runners are processes that pick up and execute jobs for GitLab. Here you can register and see your Runners for this project.
More information

You can set up as many Runners as you need to run your jobs.
Runners can be placed on separate users, servers, and even on your local machine.

Each Runner can be in one of the following states:

- active - Runner is active and can process any new jobs
- paused - Runner is paused and will not receive any new jobs

To start serving your jobs you can either add specific Runners to your project or use shared Runners

Specific Runners

Set up a specific Runner automatically

You can easily install a Runner on a Kubernetes cluster.
Learn more about Kubernetes

1. Click the button below to begin the install process by navigating to the Kubernetes page
2. Select an existing Kubernetes cluster or create a new one
3. From the Kubernetes cluster details view, install Runner from the applications list

Install Runner on Kubernetes

Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup: `https://gitlab.com/`
3. Use the following registration token during setup: `LQr-ZBsYVND11m51rToR`

Reset runners registration token

4. Start the Runner!

Shared Runners

Shared Runners on GitLab.com run in autoscale mode and are powered by Google Cloud Platform. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 2000 CI minutes per month per group for private projects. Read about all GitLab.com plans.

Enable shared Runners for this project

Available shared Runners: 15

ih9XD9p3
gitlab-docker-shared-runners-manager-03 #2072964
gitlab-org-docker

Hs8mheX5
windows-shared-runners-manager-1 #1506020
shared-windows windows windows-1809

Pipeline Code: Examples

.tmo.function.pcf.gitlab-ci.yml 3.99 KB

Edit | Web IDE | Lock | Replace | Delete

```
variables:
  PCF_BG_SCRIPT: "deploy-pcf.sh"
  GIT_TAG_SCRIPT: "git-tag.sh"
  PCF_MANIFEST_PATH: "manifest.yml"

.pcf-deploy:
    image: alpine:latest
    stage: deploy
    script:
      - if ! exists "curl" ; then installPackage "curl"; fi;
         ## Pristine Autoconfig step '
      - |
        set +ex;
        if [ "$PRISTINE_CONFIG_PCF_ENABLE" == "true" ]; then
            echo "[INFO][PRISTINE] : PRISTINE_CONFIG_PCF_ENABLE = $PRISTINE_CONFIG_PCF_ENABLE "
            deploytype=$( expr match "$PCF_BG_STRATEGY"  '\(GREEN_AUTO_SWITCH\|GREEN_DEPLOY_ONLY\|TOGGLE_BLUE_GREEN_AUTO_SWITCH\|TOGGLE_BLUE_GREEN_DEPLO
            echo "[INFO][PRISTINE] : $PCF_BG_STRATEGY in deploytype =  $deploytype"
            if [[  "$deploytype" ]]; then
              curl -Os --noproxy '*' "$CDP_SERVICE_URL/static/pristine-common.sh" && bash pristine-common.sh  -assetid="$ASSET_ID" -step="config" -appli
            fi;
        fi
        set -ex;
      - curl -Os --noproxy '*' $CDP_SERVICE_URL/static/pcf-install.sh && sh pcf-install.sh
      ## Pre deploy QCP publish
      - curl -Os --noproxy '*' $CDP_SERVICE_URL/static/qcp.sh && sh qcp.sh&
      - |
        if [ -f ${PCF_MANIFEST_PATH}.j2 ]; then
          j2 ${PCF_MANIFEST_PATH}.j2 -o ${PCF_MANIFEST_PATH}
        fi
      ## PCF deploy
      - curl -Os --noproxy '*' $CDP_SERVICE_URL/static/$PCF_BG_SCRIPT && bash $PCF_BG_SCRIPT
      ## post deploy QCP publish
      - export QCP_TYPE="post"
      - curl -Os --noproxy '*' $CDP_SERVICE_URL/static/qcp.sh && sh qcp.sh&
      ## Below Tags GIT Commit
      - |
        set +e;
        if [ -n "${POST_DEPLOY_COMMIT_TAG}" ]; then
          echo "[INFO] POST_DEPLOY_COMMIT_TAG provided is $POST_DEPLOY_COMMIT_TAG";
          {
            #download the script file and execute with file params
```

.tmo.job.pcf.gitlab-ci.yml 773 Bytes

```
include:
  # Inject the PCF function
  - project: 'tmobile/templates'
    ref: tmo/master
    file: 'gitlab-ci/.tmo.function.pcf.gitlab-ci.yml'

# Deploy to Development environment
deploy-dev:
  extends: .pcf-deploy
  rules:
    - if: $CI_COMMIT_REF_NAME =~ /^tmo\/.*$/
      when: never
    - when: on_success
  environment:
    name: dev

 # Deploy to Staging environment
deploy-stg:
  extends: .pcf-deploy
  rules:
    - if: $CI_COMMIT_REF_NAME =~ /^tmo\/.*$/
      when: manual
      allow_failure: true
    - when: never
  environment:
    name: stg

# Deploy to Production environment
deploy-prd:
  extends: .pcf-deploy
  rules:
    - if: $CI_COMMIT_REF_NAME == "tmo/master"
      when: manual
      allow_failure: true
    - when: never
  environment:
    name: prd
```

Scripts: Examples

Mandatory Parameters

PCF Variables: It is highly recommended to store these variables at *Settings > CI/CD > Variables* of the project or sub-group scoped to all/each environment.

- `CF_USERNAME` : provide the cf username
- `CF_PASSWORD` : provide the cf password
- `PCF_API` : api url e.g. api.sys.px-npe01.cf.t-mobile.com
- `DOMAIN` : domain to put app e.g. apps.px-npe01.cf.t-mobile.com (default is apps.foundation)
- `PCF_ORG` : PCF Org Name
- `PCF_SPACE` : PCF Space Name

Optional Parameters

- `PCF_MANIFEST_PATH` : PCF Manifest Path e.g. ./manifest.yml. Also, supports j2 templates (file name: manifest.yml.j2 but `PCF_MANIFEST_PATH` : manifest.yml). See Additional Capabilities for further details.
- `CF_BASE64_PASSWORD` : base 64 encoded password - Can be used instead of CF_PASSWORD if CF_PASSWORD has characters that can't be masked.
- `PCF_APP_NAME` : Can be used if you want to give a different PCF app name instead of the project name. If you don't use this variable then by default it takes the name the project ($CI_PROJECT_NAME).
- `PCF_APP_NAME_EXT` : extends the application name and route, if needed e.g. `PCF_APP_NAME_EXT=fcs` will add `-fcs` to the `PCF_APP_NAME:frontline` , which makes route: `frontline-fcs.apps.sys.px-npe01.cf.t-mobile.com`
- `PCF_ROUTE_HOSTNAME` : extends the route only, if needed e.g. `PCF_APP_NAME=frontline` , `PCF_ROUTE_HOSTNAME=frontline-qlab02` or `PCF_ROUTE_HOSTNAME=${PCF_APP_NAME}-qlab02` or `PCF_ROUTE_HOSTNAME=${PCF_APP_NAME}-${PCF_SPACE}` which makes the route: `frontline-qlab02.apps.sys.px-npe01.cf.t-mobile.com` (`$PCF_ROUTE_HOSTNAME` . `$DOMAIN`)
- `RESTAGE_APP` : restages the green app after the default route has been assigned. Needed for apps that use Eureka/service registry. e.g `RESTAGE_APP: "TRUE"`
- `POST_DEPLOY_JFROG_ARTIFACT_TAG` : Comma seperated key value pair of the tags to be applied to the jfrog artifact (e.g.:- 'LEVEL=ReleaseCandidate,TEST_STATUS=QA'). This is same as jfrog artifact tagging performed using jfrog-artifact CI Template
- `POST_DEPLOY_COMMIT_TAG` : same restrictions as vanilla git tagging labels. (e.g.:- '1.0.0-dev', '1.0.0'). This is same as GIT tagging performed using GIT Tag CI Template
- `POST_DEPLOY_SLACK_CHANNEL_WEBHOOK` : Slack Incoming Webhook URL. If provided, a default slack message will be posted to the Slack Channel after successful deployment. This is same as Slack Notify CI Stage's `SLACK_CHANNEL_WEBHOOK`
- `PCF_CLI_VERSION` : If you want to hard code the PCF CF CLI version, please put in a version number here to be downloaded on the fly. (i.e. 6.49.0)
- `PCF_ROUTE_PATH` : extends the path of the route, if needed e.g. `PCF_ROUTE_PATH=api` which makes the route `frontline-qlab02.apps.sys.px-npe01.cf.t-mobile.com/api` . In case the `DOMAIN` variable contains multiple domains, the path will be applied to all.
- `AUTOSCALER_MANIFEST` : creates and enables auto-scaling based on rules defined in the auto-scaler manifest file. e.g., `autoscaler-manifest.yml` Below is a example of a manifest file used for auto-scaling based on HTTP Lattency. The threshold values are in milliseconds. For more information and options on auto-scaling, refer to Using the App Autoscaler CLI

```
998     renameApp blue green            # Rename blue (n-1) app to green
999     startApp green                  # Start the green app
1000    scalePCFApp green "1"           # Ensuring atleast 1 instance is set before healthcheck
1001    assignAppRoute green green      # Assign green route to green App
1002    assignAppRoute green default    # Assign default route to green app
1003    optionalAppHealthCheck "green"
1004    # Scale up the green app
1005    scalePCFApp green
1006    removeAppRoute default default  # Remove default route from default app
1007    scalePCFApp default "0"         # Scale down the default app (n)
1008    stopApp default                 # Stop the default App
1009    removeApp default               # Remove/Delete the Bad default app
1010    renameApp green default         # Change green app (n-1) to default app.
1011    removeAppRoute default green    # Remove the green route from the default app
1012}
1013
1014case $PCF_BG_STRATEGY in
1015    MANUAL_DEPLOY)
1016       echo "[INFO] Deploying with cf push from $PCF_MANIFEST_PATH"
1017       eval "cf push ${CF_PUSH_OPTIONS}"
1018       # Don't undo_bluegreen because we don't know the PCF_APP_NAME
1019       ;;
1020    GREEN_AUTO_SWITCH)
1021       green_deploy
1022       removeApp blue
1023       move_green_to_default_route
1024       renameApp default blue #Rename the existing default running app as blue (for backup)
1025       renameApp green default # Change green app to default app.
1026       optionallyRestageApp default # needed after route changes, if needed
1027       [ "$KEEP_OLD_APP_RUNNING" == "false" ] && stop_blue_from_default_route
1028       #Final State: app-default route has 2 apps: app(running), app-blue(stopped).
1029       ;;
1030    GREEN_DEPLOY_ONLY)
1031       green_deploy
1032              #Final State: app-green route has 1 app: app-green.
1033       ;;
1034    SWITCH_GREEN)
1035       validateAppReadiness "green" # Before switching ensure -green exists
1036       removeApp blue
1037       move_green_to_default_route
1038       renameApp default blue #Rename the existing default running app as blue (for backup)
1039       renameApp green default # Change green app to default app.
```